Informatics in Education, 2013, Vol. 12, No. 1, 43–58

Experience in Teaching OOAD to Various Students

Danijela BOBERIĆ-KRSTIĆEV, Danijela TEŠENDIĆ
Faculty of Sciences, University of Novi Sad, Serbia
e-mail: dboberic@uns.ac.rs, tesendic@uns.ac.rs,

Received: September 2012

Abstract. The paper elaborates on experiences and lessons learned from the course on object-oriented analyses and design at the Faculty of Sciences, Novi Sad. The course on OOAD is taught to students of computer science and to the students of mathematical programme. Conclusions made in this paper are based on results of students' assignments as well as results of conducted survey. In the paper we identify a set of issues concerning teaching modelling and UML. It is noticed that difficulties in mastering OOAD arise primarily from the absence of appropriate real case studies from the field of designing information systems. In order to overcome this problem, students worked on their own homework projects which include all phases of software development. Concerning the results of survey it is noticed that OOAD course should be taught in different manners regarding previous knowledge of students. Suggestions how to teach OOAD to students of computer science and to students of other programmes are given in this paper.

Keywords: OOAD, UML, teaching, modelling.

1. Introduction

Courses on object-oriented modelling have become part of compulsory education at a large number of world universities involved in computer science. The goal of such courses is that students learn about basic concepts of object-oriented analysis and design (OOAD) as well as the basic concepts of the Unified Modelling Language (UML). UML (2013) is a language that helps to visualize, design and document models of software systems. UML includes a set of graphic notation used to represent some concepts of object-oriented paradigm. UML provides several diagram types that can be used to view and model the software system from different perspectives and different levels of abstraction. UML diagram presents a graphic representation of a system's model. UML 2.0 defines thirteen types of diagrams, divided into three categories: structure diagrams, behaviour diagrams and interaction diagrams. Structure diagrams describe static structure of the system and they are used in documenting the software architecture. Structure diagrams include the class diagram, object diagram, component diagram, composite structure diagram, package diagram and deployment diagram. Behaviour diagrams represent behaviour of the system and they are used to describe the functionality of software systems. Behaviour diagrams include the use case diagram (used by some methodologies during requirements gathering), activity diagram and state machine diagram. Interaction diagrams describe flow of control and data among objects in the system and include the sequence diagram,

communication diagram, timing diagram, and interaction overview diagram. One characteristic of UML is that it is methodology-independent. UML can be used to describe system specification regardless of the methodology that is used to perform analysis and design.

To make students ready for OOAD, its principles and applications need to be taught and incorporated in university curricula. Modelling is a means for dealing with the complexity and it supports way of constructing an abstraction of a system, which pays attention on interesting aspects of system and ignores irrelevant details. By learning concepts of OOAD, students should understand why modelling is so important and why just programming skills are not sufficient. Very often students think that all those UML diagrams are useless and serve as a documentation which no one reads, so courses dealing with object oriented modelling should focus not only on UML notation, but also on placing modelling in appropriate contexts and development of abstract thinking in students.

In this paper we presented how those ideas are implemented in the concrete university course. We identified a target reader of this paper as a teacher and practitioner who wants to teach UML based software development in a modern, proper and effective way, to design and to deliver an appropriate course. The presentation of this paper proceeds in 6 sections. At the beginning of the paper we gave brief overview of recent research in the field of teaching OOAD. We continued by laying out the course "Information systems" which deals with object-oriented modelling and is held at the Faculty of Sciences, Serbia. At the course, students learn to model information systems using UML. In Section 4, we presented results of the survey regarding quality of teaching process of this course and we described our experience gained in working with students in the past few years. Some particular problems with which students encountered in mastering UML are described. In Section 5, based on our experience and according to ongoing research in teaching OOAD, we made proposal of two different approaches in teaching OOAD to student with different previous knowledge. This proposal could be taken into consideration when teaching OOAD as a university course.

2. Related Work

A number of OOAD and UML retraining courses are being developed and delivered. Issues concerning teaching OOAD are subject matter of many papers and conferences. For instance, the Educator's Symposium at the MoDELS conference (EduSymp, 2010) takes place every year and is intended as a forum in which educators and trainers can meet to discuss pedagogy, use of technology, and share their experience pertaining to teaching modelling techniques.

Main topics discussed in the papers concerning OOAD teaching relate to issues such as complexity of UML, connection between object-oriented programming (OOP) and OOAD and teaching UML to various audiences. Wrycza and Marcinkowski (2007) believe that UML 2.0 became overwhelmed with different types of diagrams, and in the paper they made proposal of using Light Version of UML while teaching OOAD. According to survey made in that paper, students identified use case diagram, class diagram,

activity diagram and sequence diagram as the most useful diagrams for modelling essential system aspects.

However, UML should not be taught separately. Debuse and Stiller (2008) highlighted close relationship between OOAD and OOP. They suggested that integration of OOAD and OOP concepts and assignments across university courses in OOAD and OOP could improve students' productivity. Also, Guthrie (2004) discussed whether OOP should be taught before OOAD or otherwise. It is concluded that students should be taught programming first, followed by the system analysis course because frequent comparisons of software design artefacts to final code improve students' ability to create good software designs. Contrary to this opinion, Rivera-Lopez *et al.* (2009) involved in teaching the foundations of programming think that previous knowledge of UML concepts could be useful in better understanding OOP. On the other hand, Starrett (2007) believes that it is feasible to teach UML without previous knowledge of programming. He described experience in teaching modelling at the high school.

In addition, various audiences demand different approach in teaching UML. Moisan and Rigault (2010) described their experience in teaching UML to students, professional software developers and non-developers. Acceptance of UML varies with the origin of the attendees. It is usually better for students than for professionals, especially because students do not have yet any habits in software development and the graphical nature of the models is rather appealing. Based on their experience authors gave some practical guidelines how to teach UML to different audience. The authors particularly advocate for teaching UML as a university course.

Different research papers describe experience in teaching UML at universities. In one of those papers is described experience of teaching UML at the University of Texas at Dallas (Cooper *et al.*, 2005). In that paper, authors described usage of UML in different graduate-level software engineering courses. They explained that their course dealing with OOAD includes two examinations and one multi-phase course project. Students worked in teams of approximately 3 people and all teams had to model and implement the same system. Project consisted of two phases. In the first phase student had to deliver detailed use-case model, analysis class model and GUI interface of the system. In the second phase students had to submit implementation of working system. Also, there are different approaches in teaching UML. Labiche (2009) reported his experiences in holding course on Software Engineering at Carleton University in Canada. Students already had been introduced to UML notation and goal of the course was to heighten students' awareness of the problem of the well-formedness of UML diagrams. In order to achieve that, students were provided with analysis document which consisted of use case diagrams, use case descriptions, class diagrams and interaction diagrams. Students had a task to find as many as possible inconsistencies which are intentionally seeded in document by instructor. Author concluded that through this approach students improved their ability to apply UML and became aware of connection among different diagrams which describe the same system.

In the following sections, it is described experience which is gained by holding course on OOAD at the Faculty of Sciences in Novi Sad, Serbia. Object-oriented modelling

concept has been taught within the context of a course titled "Information systems" which is offered at the undergraduate programmes.

3. Overall Structure of the Course

The Faculty of Sciences consists of several Departments. At the undergraduate level at Department of Mathematics and Informatics, there are two mathematical programmes and one programme in computer science. Before taking course "Information systems", students of computer science programme already had courses dealing with OOP (with an introduction to the Java programming language) as well as a course on foundations of database concepts. Those courses were mandatory for students of computer science programme, but for other students those courses were elective.

The course on object-oriented modelling is organized in such way that students attend three hours of lectures and two hours of exercises on computers per week during one semester. The course is compulsory for students of computer science programme and for the students of mathematical programme this course is elective. The course has been running in its present form during the last three years. Every year approximately 120 students of which 30% are students of mathematical programmes attend the course. The course is oriented along a simplified software development process which starts with a requirements gathering phase, analysis phase and design phase. The different models and UML diagrams are employed during this process. Remaining phases of software development process are taught on another, following course where students deal with implementation of information systems. Students make just specification of one concrete information system on "Information systems" course, while on the following course they implement that system in the chosen technology.

Lectures give overview of methodologies used for information system development. Great emphasis is placed on object-oriented methodology and visual system modelling using UML. Students are introduced with different views which are used to describe the system from viewpoint of different stakeholders, such as end-users, developers and project managers. Different views are explained through the UML diagrams that are commonly used in modelling. Functionality of the system from the viewpoint of the end user is represented through the use case diagrams. Static structure of system includes class diagrams. View of system from a programmer's perspective is described using component diagrams and package diagrams. Dynamic aspects of system include the activity diagrams, communication diagrams and sequence diagrams. Topology of software components on the physical layer, as well as the physical connections between these components is represented by deployment diagrams. Also, in parallel with the adoption of the theoretical basis of UML, students master UML diagrams through the various teaching examples. Those examples are not case studies, they only place emphasise on particular view of the system which is modelled and the usage of appropriate UML diagram.

On the practical exercises students draw UML diagrams using Sybase PowerDesigner CASE tool with academic licence (Sybase, 2012). The idea of practical exercises is to

refine students' knowledge of UML obtained through the lectures. Due to the lack of time, they only work with several UML diagrams such as use case diagrams, activity diagrams, class diagrams and sequence diagrams.

During the semester, students have two tests to demonstrate knowledge of the basic concepts of these diagrams.

The first test deals with specification of system requirements using UML. Students are provided with assignments in a textual form containing description of functional and non-functional requirements which should be supported by information system. Text of the assignment intentionally contains some imprecision and uncertainties, because that is common way how an end user describes his/her own requests. According to that assignment, student should identify actors in the system, use cases, relationships among use cases and finally to create use case diagram. The remaining part of the test relates on presentation of different scenarios of single use case using activity diagrams.

The second test deals with specification of static and dynamic model of the system. The second test is comprised of use case diagram of some particular system and textual description of single use cases. Based on that, students should create analysis class model and identify three types of classes: Entity, Control and Boundary classes. Those classes are presented on the class diagram, while communication between them is presented on the sequence diagrams.

Also, at the end of course, students have to submit projects. They work in a team of up to four students and have possibility to propose the system which they want to model. That project comprises of:

- description of the functional requirements of an information system that is designed and detailed diagram of the use cases with text descriptions of individual use cases;
- an analysis class model that identifies and represents the building blocks of the system and relationships between them at analysis level;
- GUI interface model describing the major graphic user interfaces.

4. Teaching Experience

UML 2.0 describes 13 different types of diagrams which are used for modelling different aspects of information system. It is necessary to make balance between theory and practical usage of some kind of diagrams. Inside of course "Information systems" we have introduced students with all 13 types of diagrams, but we have insisted only on several types. We have chosen use case diagram, activity diagram, class diagram, sequence diagram and component diagram to be practiced because we think that they are the most useful. Wrycza and Marcinkowski (2007) made similar conclusion. Those diagrams were also involved in the students' assessments. Considerations presented in this section are based on analysis of assessments' results and on an anonymous survey conducted on students who took the course "Information systems".

4.1. *Results of Survey*

Results presented in this section are based on the survey which is conducted during last two years. The reason for this survey was to get anonymous feedback from the students regarding quality of teaching process. Our main objectives were to estimate previous knowledge of students, to determine the level of usefulness and acceptance of particular UML diagrams and to value importance of team work. 234 students participated in this survey. 84 of them are students of mathematical programme and 150 are students of computer science programme. The survey was paper-based and was comprised of eleven multi-choice questions.

The first aim of this survey was to estimate previous knowledge of the students who attended the course "Information systems". Survey showed that 88% of students of computer science have satisfying knowledge of object-oriented programming language (OOP) which was expected because those students had obligatory course dealing with object oriented concepts. On the other hand, 80% of students of mathematics said that they have no knowledge of OOP. In the Fig. 1, results of survey relating to programming languages and the level of their knowledge are presented. As it can be seen, all students of computer science have knowledge of Java, and about one half of them know C#. All students of computer science know at least one programming language. On the other hand, 68 students of mathematics said that they do not know any programming language.

In addition, we wanted to know what is their knowledge regarding databases. Students of computer sciences had mandatory course relating to databases so survey showed that almost all students have satisfying knowledge. When it comes to students of mathematics, course relating to databases was elective for them and survey showed that just 30% of them have satisfying knowledge of databases. We have to distance from this judgment because we did not explicitly test student's knowledge, they estimated their knowledge by their own opinion.

The next question on what we wanted to get answer through this survey regards level of acceptance of particular UML diagrams. Those findings are presented in the Figs. 2 and 3 for students of computer science and students of mathematics, respectively. The level

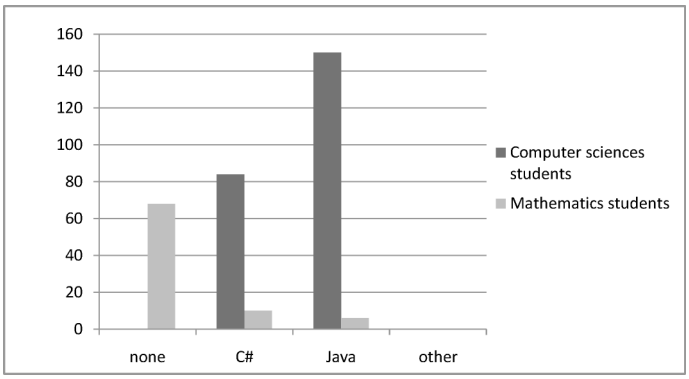

Fig. 1. Knowledge of programming languages.

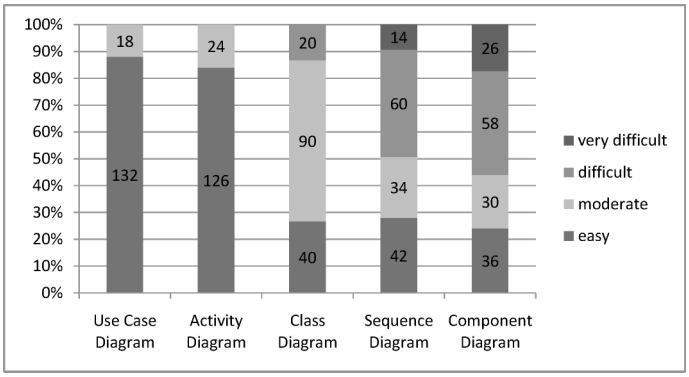

Fig. 2. Students of computer science – level of acceptance of UML diagrams.

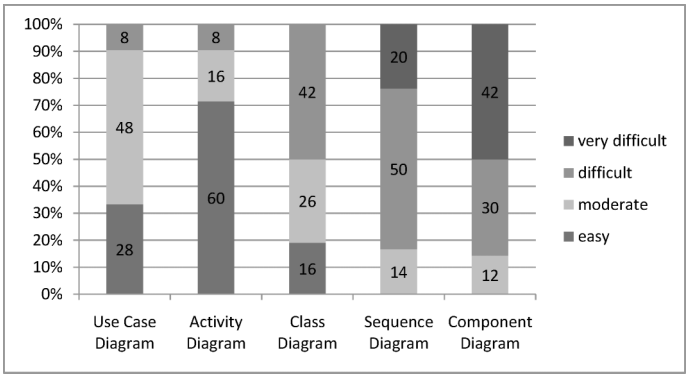

Fig. 3. Students of mathematics – level of acceptance of UML diagrams.

of acceptance is ranged from easy to very difficult. As it can be seen, the most of students of computer science had no problems with UML notation relating to use case diagrams. More than half of students of mathematics said that use case diagrams were moderately difficult. The activity diagrams were easily accepted by students of both programmes and we think that this result is connected with the fact that they all probably had some experience in algorithm design. Mastering concepts of class diagram was moderately difficult for students of computer science programme, while students of mathematics had more difficulties to understand complex concepts of class diagrams. Acquiring concepts of sequence diagram was difficult for 50% of students of computer science, while on the other hand almost all students of mathematics said that sequence diagrams are difficult or even very difficult. Students of both programmes regarded component diagram as the most difficult diagram and we think that this is particularly because they were not able to perceive different functional blocks of the system.

When it comes to usefulness of particular diagrams, we took into consideration only opinion of those students who had some experience in the software development process. About 40% of students of computer science declared that they have been involved in the

software development process. About 50% of them said the system modelling has important role in development process, while other couldn't estimate importance of modelling or said that it has little importance. None of students of mathematics said that have any experience in that field. Those students who had some insight in software development regarded use case and class diagrams as the most useful, while their opinion about other diagrams was divided and nothing can be concluded from it.

One of students' assignments in this course was to submit a project which consists of specification of one particular information system. It is supposed that this assignment is done as a team work of up to four students. Through this survey we wanted to get feedback regarding this assignment. We asked students what they think about usefulness of team work, do they have problems to work in team and does the work on the project help them to better understand UML concepts. Generally speaking students of both programmes did not have problems to work in team, however, students of mathematics regarded team work more useful than students of computer science. For instance, 80% of students of mathematics regarded team work as very useful and only 61% of students of computer science agree with that statement. On the other hand, it is very interesting that students of computer science valued work on the project as very helpful in the process of understanding UML, while students of mathematics stated that the exercises using CASE tool were the most helpful and just few of them said that work on the project was helpful too.

4.2. *Analysis of Students' Assessment*

According to students' assessments performed during the last few years, some conclusions about complexity to use different diagrams are made. In this section, we will discuss some common mistakes which students made in their assessments.

Although most of the students said that they didn't have problems to accept use case diagrams, some problems regarding functional decomposition of system are noticed. There was a problem with identifying use cases and determining the right level of use case granularity. Namely, students tended to derive smaller, finer grained use cases which represent atomic functionalities. This approach results in losing idea of main system functionalities and relations among them. Use case diagrams became overwhelmed and hard to follow. Students who created small use cases more often used include relationship. On the other side, some students had tendency to create large, more abstract use cases, which leads to ambiguous use case boundaries and problems in understanding of system. In addition, the next problem relates to students' tendency to equal use cases with specification of user interface. In other words, a number of students regarded identification of use cases as defining screens of the user interface and for every button on the screen they defined new use case, which leads to very fine grained use case model. Also, it is noticed that there was problem relating to identification of relationships between use cases. During the use case modelling it is possible to identify include and extend relationships and in some situations, students couldn't decide whether to use include or extend relationship.

As we know, particular use case can be realized through the different scenarios. Each scenario contains activities which should be performed, as well as objects which are re-

sponsible for their execution. One part of students' assignment was to describe realization of use case using activity diagram. Students were provided with textual description of several scenarios regarding one use case, and they had to present those scenarios through the single activity diagram. Generally, students who were familiar with algorithm design accepted activity diagrams easily. However, some problems were noticed. For example, some students didn't understand that those scenarios relate to one use case, so they create separate diagrams for each scenario.

Mastering concepts of class diagram was hard for students of both programmes. One of the major problems was creation of analysis class model. Analysis class model consists of three types of classes: Entity, Control and Boundary classes (Jacobson *et al.*, 1999). Entity classes represent the persistent layer of the system. Boundary classes represent presentation layer of the system, precisely, they support interaction between end user and system. Control classes are responsible for implementing business logic of the system and they are intermediary between boundary and entity classes.

Students usually have some previous knowledge of two tier architecture and by creating analyses class model they are introducing with three tier architecture. Adopting of that new concept was difficult for them. Mainly they only identified entity and boundary classes while failed to observe control classes. So, operations which should belong to control classes were assigned to boundary classes. Result of this is that students created methods of boundary classes which directly call methods of entity classes. However, there were some cases where students ignored boundary classes and created only control and entity classes. Reason for this lays in the fact that they didn't understand role of boundary classes. We are dealing with this problem by drawing sketches and mock-ups of user interface and based on that we create boundary classes and make connections between user interface actions and boundary classes' methods.

In addition, students who have previous knowledge of database modelling had difficulties in accepting concept of object-oriented model. They avoided using many-to-many associations, aggregation, composition, association classes as well as interfaces and abstract classes. Namely, they considered entity classes as a database tables instead of regarding them as layer which communicates with database. Also, as a result of that they didn't conceive the value of using operations in the entity classes. Generally speaking, it is observed that students have a little or no experience in object-oriented programming and because of that they cannot comprehend how some particular model will be implemented. It was very hard to explain to students that, for example, many-to-many association between two classes should be implemented in such way that one class has an attribute which is a collection of objects of the second class and otherwise.

Students from both programmes had problems with the adoption of sequence diagrams. Consequence of low comprehension of class diagrams resulted in problems with adoption of sequence diagrams. Sequence diagrams may be the best indicators of the level of acceptance of class diagrams and three tier architecture. It is noticed that even though students created correct class diagrams and it seemed that they understood the concepts, they still created bad solution of related sequence diagrams. Sometimes, students do not even seem to know that the different kinds of UML diagrams, such as sequence and class

diagrams, are related to one another although each of them addresses a particular aspect of the same system. The main problem lays in the fact that students only learn notations of particular diagrams and see diagrams isolated from each other.

5. Teaching OOAD to Various Audience

According to common issues which were noticed during the teaching OOAD at the Faculty of Science and according to results of the survey, in this section are given suggestions how to teach OOAD to undergraduate students. It was concluded, similarly to Moisan and Rigault (2010), that same OOAD course is not suitable for all students. So, in this section we suggested two different courses, one for students of computer science and one for students of non computer science programmes.

5.1. *Teaching OOAD to Students of Computer Science*

Taking into account that students of computer science, after graduation, can work in any field of IT and thereby may have different roles in the development of software, it is preferable that on the faculty they acquire basic knowledge and skills needed to execute each of these roles. Therefore, we believe that the students have to pass through all phases of software development.

One way that students perceive the whole process of software development is to include the analysis of complete case studies in the course. However, the main reason for the difficulty in mastering object-oriented analysis and design results primarily from the absence of appropriate real case studies from the field of designing information systems. In fact, in most literature dealing with the object-oriented modelling process individual UML diagrams are analysed and for their clarification are mainly used trivial examples that are rarely from the field of information systems. Result of using such literature, is that students learn only UML notation, but they still don't see the applicability of this in practice and in real systems. One good example of how object-oriented modelling should be taught is given in the book (Bruegge and Dutoit, 2009) where an example of real case study called Arena is used for the process of teaching OOAD. Arena is a multi-user, web based system for organizing and conducting tournament. Using this example which is real and sufficiently complex, author goes through all the stages of designing information system. However, Arena case study is not complete because of limited space of the book so, for students, it is still not appropriate example to understand the process of systems modelling.

We believe that the results would be better if students have at their disposal a number of complete case studies describing real information systems. The need for real, not academic case study is also highlighted in the work of Moisan and Rigault (2010). In order to meet those needs, we created case study which describes modelling of library management system. In that case study we started with functional requirements of the system and modelling of use case diagrams, then we described the architecture of the system using component diagrams, and then each component was described using class and sequence

diagrams. At the end of the case study we made a proposal of graphical user interface of the system.

We also believe that theory and practice must be connected, so we based our case study on a real library management system called BISIS. System BISIS has been developed at our faculty and we are included in its development. During the development of system, number of scientific papers was published. Currently, the fourth version of the system is in use. Details of modelling and implementation of this version are given in the papers (Tešendić *et al.*, 2009; Boberić and Surla, 2009; Dimić and Surla, 2009). The system is developed in Java environment, using open source software solution. This system is used in 42 libraries of Republic of Serbia. We assumed that our experience gained through developing of the system could be used to create complex and quality case study. The main advantage of this case study is that when we finish modelling of the system we can show students the real implementation of the system on which our case study is based. Through this case study students can make connection between modelling and implementation of system and they can understand why modelling takes so important role in the process of the information system development.

As we mentioned, existence of appropriate case study is important to teach modelling because it includes all phases of system developing. Similar to that, it is very important that students independently go through all phases of development. The best way to do that is that students work on their own homework projects. It is noticed that students have better results on tests after they finish their projects. By working on projects they gain better insight in system modelling. This is confirmed by the fact that students from previous years, for who was not mandatory to submit projects, demonstrated lower level of comprehension of UML. After we included students' projects as mandatory part of the course, students had possibility to choose which system they want to model in their projects. They had to define scope of the problem and all functional requirements of the system. However, our experience showed that level of acceptance of OOAD concepts would be better, if they had to work on the problems which were predefined. By allowing students to choose subject matter of the projects, very often we got projects which were not complex enough and on such projects they couldn't deal with advanced concepts of OOAD. Because of that we decided to play role of end user and we defined all functional and non-functional requirements of system which students have to model.

Furthermore, it is observed that students create UML diagrams just for the sake of modelling. They do not make connection between models and their implementation. They cannot comprehend how some particular model will be implemented and it seems that they even don't realize that model is base for future implementation of the system.

In order to overcome those problems, implementation should be important part of every course which deals with OOAD. Taking into account that modelling and implementations are two sides of the same coin, they should not be taught separately. In this aspect we agree with the authors Debuse and Stiller (2008) and Guthrie (2004). We consider that students must acquire appropriate programming skills in implementing UML models. Teacher should pay additional attention to that problem of implementation of UML models in the concrete OO programming languages. By showing implementation

of particular UML model students will better understand elements of UML which are used. Students should be introduced with way how some particular concepts of UML are mapped into a concepts of target object oriented programming language. For example, they should see that relationships between UML classes will be implemented in such manner that each implementation class contains a reference to related class according UML association. According to these facts we cannot agree with Starrett (2007) who stated that knowledge of OOP is not prerequisite for learning OOAD.

Although modern CASE tools provide the ability to generate programming code based on the model, that code is very often limited to skeletal code which even do not contain references between classes. Generated code by case tools is not sufficient to understand all concepts of UML, so students should implement their models by themselves using their own programming skills. The authors of this paper think that one single course is not enough to learn OOAD and to make implementation of obtained models, which is opposite to idea of Cooper *et al.* (2005) where students work on project which includes both modelling and system implementation. At our faculty, students after finishing course dealing with OOAD can attend a course dealing with developing enterprise applications in Java environment. Considering the fact that students must create concrete applications on that course, our idea is that they should make applications which will be based on the models obtained through the OOAD course. In that way they will complete the whole process of information system development, starting from the modelling to concrete implementation.

5.2. *Teaching OOAD to Students of Other Programmes*

It was noticed that students of mathematical programme have problems in mastering material from the course. In fact, as these students did not have subjects such as databases or object-oriented programming, they have more problems with acquiring the basic concepts and even the terminology of UML. By analyzing the tests that were conducted during the semester, we came to the conclusion that students of computer science and mathematics are equally good at gathering user requirements and modelling functionality of system using use case diagrams, while significant differences can be seen when it comes to modelling static system structures and dynamic behaviour using class and sequence diagrams. Students of mathematical programme have difficulty in mastering concepts such as inheritance, polymorphism, abstract classes and interfaces, which can be attributed to the lack of prior knowledge of object-oriented programming.

Although Starret (2007) stated that to overcome the modelling techniques should not require any prior knowledge, we believe that students with OOP knowledge adopt OOAD more easily. This implies that we should have different approaches in teaching students with different knowledge. In addition, we should have different courses which will be adjusted to the previous knowledge of the students as well as to the students' needs for OOAD.

Based on our experience we came to conclusion that material presented inside the course "Information system" is too complex for non-IT students, primarily because of

their poor previous knowledge. Therefore, we believe that for non-IT students the material should be divided into several courses.

Taking into account that students of mathematics had no difficulties in mastering material related to the modeling of system functionality and taking into account that process of system development comprises various activities and includes many different roles, we believe that successful students of mathematics, or students from any programme which includes topics related to software engineering, could find their place in the development of information systems. So, it is highly likely that those students will have different roles in comparison with students of computer science programme. According to all previously mentioned, our idea is to design course for non-IT students which will put stress on requirements engineering.

Generally speaking, requirements engineering is a branch of software engineering which includes activities like eliciting requirements, modelling and analysing requirements, communicating requirements, agreeing requirements and evolving requirements. During the process of requirement engineering, for example, it is necessary to identify stakeholders and their needs and document them in a suitable form. However stakeholders may have different goals which may not be explicit or may be difficult to articulate, because of that, students of non-IT should be taught how to deal with those situations. So, for them it is very important to master those UML diagrams which are essential for requirements modelling and just to acquire basic concepts of other UML diagrams to be able to communicate with other team members.

According to our opinion, course dealing with requirements engineering besides modelling skills should include interviewing and groupware skills for requirements elicitation, and writing skills for specifying requirements. Also, the authors of this paper propose that one of the tasks should be design of requirement specifications for a particular system and the teacher should play the role of stakeholders. The role playing would give students a greater appreciation of the range of issues and problems associated with requirements engineering in real settings. Students could employ use case and activity diagrams for the purpose of analysis and modelling requirements. In order to document requirements and manage their changes students should use some kind of requirement management tool. For example, they also can use Power Designer which supports requirement management.

Proposed course could be the first course at which the non-IT students learn about the process of software development. Other activities of the software development process could be handled in the advanced courses dealing with software engineering.

6. Conclusion

Modelling is central part in doing and learning object-oriented development. Course of OOAD should be organized in such manner that students can understand why modelling is so important and why just programming skills are not sufficient. UML, as an OOAD notation, seems to help students to better appreciate the necessity of analysis and design of systems. This paper reports the recent experiences in teaching OOAD at the Faculty of

Sciences, University of Novi Sad, Serbia. We described the course dealing with modelling of information system using UML. This course is attended by students of mathematical and computer science programme. Observations made in this paper can be valuable to those who are involved in teaching and designing courses dealing with OOAD.

In the paper we identified a set of issues within teaching modelling and UML. First of all, it became clear that lack of non-academic case studies influences the quality of knowledge acquired by students. In order to overcome this problem, we created case study which describes modelling of library management system. Our case study is based on real library management system called BISIS which is used in 42 libraries of Republic of Serbia. The main advantage of this case study is that students can see the real implementation of the system on which our case study is based. Based on our experience we concluded that students need to be actively involved in the learning process and must be able to apply a concept while it is being taught. Because of that we insisted on students' projects which should be done as homework. It is noticed that after submitting their projects students have better overview of OOAD.

Also, concerning results of students' assignments as well as results of conducted survey, we noticed that it is necessary to pay additional attention when teaching OOAD to various audiences. The authors of paper proposed to have two different courses which will deal with concrete aspects of software development. For students of computer science it is essential to go through all activities in the process of system development and because of that course designed for them should include team work on a complex project starting from requirements gathering to implementation. On the other hand, non-IT students should just be introduced with the process of software development and course designed for them should put stress on requirements engineering.

Acknowledgments. The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. 174023: "Intelligent techniques and their integration into wide-spectrum decision support".

References

Boberić, D., Surla, D. (2009). XML editor for search and retrieval of bibliographic records in the Z39.50 standard. *The Electronic Library*, 27(3), 474–495.

Bruegge, B., Dutoit, A.H. (2009). *Object-Oriented Software Engineering Using UML, Patterns, and Java*. 3rd edn., Prentice Hall.

Cooper, K., Dong, J., Zhang, K., Chung, L. (2005). Teaching experiences with UML at the University of Texas at Dallas. In: *Educators Symposium of the 8th International Conference on Model Driven Engineering Languages and Systems*, 1–8.

Debuse, J.C.W., Stiller, T. (2008). Technologies and strategies for integrating object-oriented analysis and design education with programming. In: *19th Australian Conference on Software Engineering*, 97–103.

Dimić, B., Surla, D. (2009). XML Editor for UNIMARC and MARC21 cataloguing. *The Electronic Library*, 27(3), 509–528.

EduSymp @ MODELS (2010). In: *6th Educators' Symposium: Software Modeling in Education*, Oslo, Norway.

Guthrie, R.W. (2004). Integrating programming and systems analysis course content: resolving the chicken-or-the-egg dilemma in introductory IS courses. *Information Systems Education Journal*, 2(1).

Jacobson, I., Booch, G., Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley, Reading, MA.

Labiche, Y. (2009). The UML is more than boxes and lines. In: Chaudron, M.R.V. (ed.), *Models in Software Engineering*. Springer, Heidelberg, 5421(1), 375–386.

Moisan, S., Rigault, J. (2010). Teaching object-oriented modeling and UML to various audiences. In: *MODELS 2009 Workshops*, 40–54.

Rivera-Lopez, R., Rivera-Lopez, E., Rodriguez-Leon, A. (2009). Another approach for the teaching of the foundations of programming using UML and Java. In: *Proceedings of the 3rd WSEAS International Conference on Computer Engineering and Applications (CEA'09)*, 279–283.

Starrett, C. (2007). Teaching UML modeling before programming at the high school level. In: *Seventh IEEE International Conference on Advanced Learning Technologies.*

Sybase PowerDesigner.
`http://www.sybase.com/products/modelingdevelopment/powerdesigner` (accessed Jan. 2013).

Tešendić, D., Milosavljević, B., Surla, D. (2009). A library circulation system for city and special libraries. *The Electronic Library*, 27(1), 162–186.

UML, The Unified Modeling Language.
`http://www.uml.org/` (accessed Jan. 2013).

Wrycza, S., Marcinkowski, B. (2007). A light version of UML 2: survey and outcomes. In: *Proceedings of the 2007 Computer Science and IT Education Conference*, 739–749.

D. Boberić-Krstićev has worked at the Department of Mathematics and Informatics, Faculty of Science, Novi Sad on the position of research assistant from 2007 to 2010. Mrs. Boberić-Krstićev received her bachelor degree in 2005 and master degree in 2007 both in computer science from the University of Novi Sad, Faculty of Science. In 2010 she received her PhD degree and became an assistant professor. She gives lectures on the subject information systems at the Department of Mathematics and Informatics. She is an active participant on the projects supported by the Ministry of Education and Science of the Republic of Serbia and she has published 11 papers related to the development of the library information systems.

D. Tešendić has worked at the Department of Mathematics and Informatics, Faculty of Science, Novi Sad on the position of research assistant from 2005 to 2010. Ms. Tešendić received her bachelor degree in 2004 and master degree in 2007 both in computer science from the University of Novi Sad, Faculty of Science. In 2010 she received her PhD degree and became an assistant professor. She gives lectures on the subject computer networks and information systems at the Department of Mathematics and Informatics. She is an active participant on the projects supported by the Ministry of Education and Science of the Republic of Serbia and she has published 10 papers related to the development of the library information systems.

Objektinės analizės ir dizaino dėstymo įvairių lygių studentams patirtis

Danijela BOBERIĆ-KRSTIĆEV, Danijela TEŠENDIĆ

Straipsnyje pristatoma objektinio programavimo dėstymo Novi Sado universitete patirtis. Objektinio programavimo kursas dėstytas kompiuterių mokslo ir matematikos specialybių studentams. Straipsnyje pateikiamos išvados grindžiamos studentų rezultatų vertinimais ir apklausa. Pastebėtos problemos dėstant modeliavimą ir UML atskleidžia, kad pagrindiniai sunkumai dėstant objektinės analizės ir dizaino (OOAD) kursą kyla dėl realių pavyzdžių, taikomų projektuojant informacines sistemas, stokos. Siekiant eliminuoti minėtas problemas studentai turėjo įgyvendinti projektus, apimančius visus programinės įrangos kūrimo ciklus. Remiantis apklausų rezultatais daroma išvada, kad OOAD kursas turi būti dėstomas skirtingais būdais, atsižvelgus į anksčiau studentų įgytas žinias. Straipsnyje pateikiami siūlymai, kaip dėstyti OOAD kompiuterių mokslo ir kitų specialybių studentams.